# Interview Questions

Due date: by the end of the recitation session

## Introduction / Motivation

Whether you are majoring or minoring in computer science, you may be going for interviews that ask technical questions. In order to answer these types of questions you need to:

- understand the question that is being asked (which means know the vocabulary that is often used in describing different data structures, algorithms and language features)
- figure out the solution to the problem or at least figure out the ”right direction” for solving the problem
- present your solution in a way that is understandable to others (i.e., be able to communicate what it is that you would do to solve the problem)

The following questions come from actual interviews that our past students were asked in interviews with different companies when they applied for jobs and/or internships. The list below contains several different questions. At this point in the course, you know enough theory that you should be able to answer these questions.

The objective for this recitation is NOT to answer all of them, but to answer a few COMPLETELY - describe a quick solution (the one that you can come up with quickly), think about its efficiency: could it be done with a better time performance and/or using less memory, describe the improvements that you come up with.

You should pick two-three problems to solve (it may be more interesting and beneficial to you if you pick harder problems for which you do not know the answer immediately). You could try to find solutions on the web, but this is not the point of this activity. Try to put yourself in a situation where it is only you and your group members and your collective brain power. Your grade here does not depend much on how quick or correct the solution is, but when you are at a real interview, it might.

**Worksheet**: `https://goo.gl/xQUWId`

## Questions:

1. Add two binary strings (i.e., write an algorithm that given two binary strings computes and returns a binary string that is the sum).
   ```
   “1001” + “1”  -->  “1010”
   “1110” + “11”  -->  “10001”
   ```
2. Given a string of open and close parenthesis and an index of an open parenthesis, find the index of the matching closing parenthesis. Assume no other characters in the expression and that the expression is valid (i.e., all open parentheses are closed and they all appear in the correct order).
   ```
   ((())()), 1    -->   4
   ((((()(()))))), 2  -->  9
   ```
   Follow-up: rewrite your solution so that it works even if the expression is not valid (it should return -1 in such case).
3. You’re given 2 integers, x and y. Find a way to compute x raised to y using an algorithm with performance better than O(y).
4. Implement string "compression" that replaces repeated counts of the same character by a character followed by its count:
   ```
   "aaabcccccaade"  -->  "a3bc5a2de"
   ```
   If the "compressed" string is the same length as the original, the function should just return the original.
5. Given a list of unique words, find the pairs of words that, when concatenated, will form a palindrome.
   ```
   ["gab", "cat", "bag", "alpha"]   -->  [["gab", "bag"], ["bag", "gab"]]
   ```
6. Given a singly linked list, how would you find a node that is n nodes away from the last node? Think of iterative and recursive solutions. You can make only a single pass through the list. You cannot use any additional data structures.
7. Given two sorted linked lists, write code that merges them into a single sorted linked list (do not copy the content of the existing lists into a new list, just merge the two into one using the existing nodes).
8. How would you design a stack which, in addition to traditional push() and pop() operations, also provides a max() function? Your push(), pop() and max() functions should be O(1). (Make sure that your plementation of max() would still be O(1) even after pop() removes the largest element.)
9. Partition a linked list of numbers around a given value x: move all the nodes with values smaller than x before the nodes with values larger than x. Do not use any extra data structures.
10. Given an array of words (a dictionary) and a word, return an array of anagrams of the word (the anagrams should come from the given dictionary).
11. Given an array of stock prices of a company, return the highest profit you can earn from buying and selling a stock. The elements in the array represent stock prices on a given day.
    ```
    [5, 8, 15, 3, 6, 5, 15, 10]`  -->   12
    (because you would buy when the price is  3 and sell when it is  15)
    ```